Numerical Optimization Project2: Solvers Survey

Gou Boyu
ShanghaiTech University
2019533179
`gouby@shanghaitech.edu.cn`

Abstract

This is the second project of 2021 fall Numerical Optimization. In this project, I will introduce filterSQP, Ipopt, Knitro(Active&Interior) as three NLP solvers and Gurobi as an MILP solver and their comparison on performance as well.

1 filterSQP

filterSQP implements an SQP trust region algorithm with a “filter” to promote the global convergence. The filter method is developed by Fletcher and Leyffer. The idea is motivated by the aim of avoiding the need to choose penalty parameters in penalty functions or augmented Lagrangan functions and their variants.

filter is a list of pairs (f^l,h^l). A new step is accepted whenever it improves the objective or the constraint violation compared to the elements of the filter. Otherwise, the step is rejected. It is a method without penalty and has global convergence. The site of user document is:

`http://www-unix.mcs.anl.gov/~leyffer/papers/SQP_manual.pdf`

1.1 Problem formulation

filterSQP solves nonlinear programming of the following form:

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & I_x \leq x \leq u_x \\ & I_x \leq x \leq u_x \end{aligned} \tag{1}$$

To simplify the notation, in the following discussion we assume the problem is in the following form:

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & c(x) \leq 0 \end{aligned} \tag{2}$$

1.2 Introduction to filter

There are two competing aims in nonlinear programming. The first is the minimization of the objective function f and the second is the satisfaction of the constraints. Conceptually, these two conflicting aims can be written as

$$\min_x \quad f(x) \tag{3}$$

and

$$\min_x \quad h(c(x)) \tag{4}$$

where

$$h(c(x)) = ||c^+(x)||_1 = \sum_{j=1}^{m} c_j^+(x) \tag{5}$$

is the $l1$ norm of the constraint violation. Here $c_j^+(x) = \max(0, c_j)$.

A penalty function combines (4) and (5). Instead filter method views them as separate aims. The filter method keeps a list of pairs (f^l, h^l) such that no pair dominates any other. A pair (f^k, h^k) is said to dominate another pair (f^l, h^l) if and only if both $f^(k) \leq f^(l)$ and $h^(k) \leq h^(l)$.

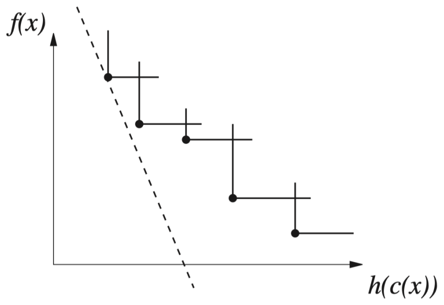


Figure 1: Example of a filter and a penalty function

The key idea is to use the filter as a criterion for accepting or rejecting a step in an SQP method. A point$(f^(k), h^(k))$ is acceptable if it is not dominated by any pair in the filter list.

Starting at $x^{(k)}$, the solution of the current QP subproblem

$$\begin{aligned} \min_x \quad & \frac{1}{2} d^T W^k d + d^T g^k \\ \text{s.t.} \quad & A^{k^T} d + c^k \leq 0 \\ & ||d||_\infty \leq \rho \end{aligned} \tag{6}$$

produces a trial step d^k and $x^{(k+1)} = x^{(k)} + d^k$. If the step is rejected by the filter, the trust-region radius ρ is reduced. Thus the usual criterion of descent in the penalty function is replaced by the filter criterion.

Algorithm 1 Basic filter SQP

```
Input: $x^0$, $\rho$, $k = 0$
while not converged do
    if the QP is infeasible then
        Find a new point $x^{(k)}$ in the restoration phase
    else
        Solve (6) for a step $d^k$ and set $x^{(k+1)} = x^k + d^k$
        if $f^{k+1}, h^{k+1}$ is acceptable to the filter then
            Accept $x^{(k+1)}$ and add $f^{k+1}, h^{k+1}$ to the filter
            Remove points dominated by $f^{k+1}, h^{k+1}$ from the filter
            Possibly increase the trust-region radius $\rho$
        else
            Reject the step (set $x^{(k+1)} = x^k$)
            Reduce the trust-region radius $\rho$
        end if
    end if
    Set $k = k + 1$
end while
```

Alg. 1 is the basic filter algorithm. In filterSQP, the QP subproblems are solved by a robust QP solver bqpd which implements a null-space active set method. The restoration phase in Alg. 1 is a strategy to deal with the case that the QP is infeasible which may caused by reducing the trust-region radius or the constraints are not consist. The detail will be talked in the following section.

1.3 Algorithm extensions

In practice, almost all the test problems can be solved by the basic filter algorithm. However, it may fail or converge slowly in some cases. So some algorithmic extensions and techniques are presented for the difficulties. The complete filterSQP combines all the following extensions and techniques into Alg. 1.

1.3.1 Second order correction step

An important property of any SQP code is that it usually exhibits second order convergence near the solution. The reason for this is that near the solution, an SQP method resembles Newton's method for solving the Kuhn-Tucker conditions and second order convergence is guaranteed under mild assumptions, as long as the unit step is accepted.

Unfortunately the use of a non-differentiable penalty function can preclude the acceptance of the unit step arbitrarily close to the solution, thereby preventing second order convergence. This is known as the Maratos effect. The difficulty can be avoided by computing a correction to the step that eliminates second order contributions of the nonlinear constraints. This is referred to as the Second Order Correction (SOC) step.

The SOC step is computed whenever $x^{(k+1)}$ is rejected by the filter in Alg. 1. For the QP defined as (6), if $x^{(k+1)}$ is rejected by the filter, a sequnce of SOC steps is performed generating a sequnce of trial points $x^{(k+1,l)}, l = 0, ...$ until one of the following holds

1. an acceptable trial point $x^{k+1,L}$ is found,
2. an infeasible QP is detected,
3. the rate of convergence of the SOC steps

$$r = \frac{h^{(k+1,l)}}{h^{(k+1,l-1)}} \tag{7}$$

   is considered to be too slow or
4. an almost feasible point with $h^{(k+1,l)} < \epsilon$ is generated (ϵ is the tolerance of the NLP solver)

In the first case the next iterate is $x^{(k+1)} = x^{(k+1,L)}$ and the filter SQP algorithm continues. In the latter three cases, the steps are rejected and the trust-region radius is reduced (4. ensures the finiteness of this process). For later use, we store the best step of this sequence of SOC steps using a penalty function estimate to rank the steps.

1.3.2 Upper bound on constraint violation

To prevent the (unlikely) situation in which a sequence of points for which $f^{k+1} < f^k$ and $h^{k+1} > h^k$ with $h^k \to \infty$ is accepted, we add an additional necessary condition for accepting a point, namely $h(c(x)) \leq u$, i.e. add an entry $(-\infty, u)$ in the filter.

1.3.3 Feasible restoration phase

In an SQP trust region method, reducing the trust-region radius will ultimately give rise to an infeasible QP subproblem if the current point is infeasible in the NLP problem.

There are various ideas are invoked when QP subproblem is infeasible. Fletcher and Leyffer suggests restoration phase as an attempt to get close to the feasible region of the NLP problem.

If an infeasible quadratic programming problem is detected, then the solver exit with a solution of the following LP

$$\begin{aligned} \min_{x} \quad & \sum_{j \in J} \nabla c_j(x_k)^T d \\ \text{s.t.} \quad & \nabla c_j(x_k)^T d + c_j(x_k) \leq 0, \qquad j \in J^{\perp} \\ & ||d||_\infty \leq \rho \end{aligned} \tag{8}$$

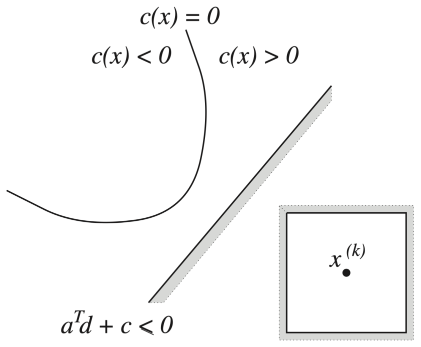


Figure 2: Infeasible QP caused by small trust-region radius

Here, J contains infeasible constraints at point $x_k : c_j(x_k) + \nabla c_j(x_k)^T d > 0 \quad j \in J$. $J^{\perp}$ is its complement, i.e. $J^{\perp} = 1, 2, ..., m \backslash J$.

(8) maintains the feasible constraints satisfied and minimize the l_1 sum of the violated constraints

The strategy in the restoration phase is to apply SQP trust-region method to the nonlinear problem:

$$\begin{aligned} \min_{x} \quad & \sum_{j \in J} c_j^+(x) \\ \text{s.t.} \quad & c_j(x) \leq 0, \qquad j \in J^{\perp} \end{aligned} \tag{9}$$

We could clearly expect to obtain an x_k that has low violation by solving NLP above. However, the difficulty is that the partition sets $J^{\perp}$ and J probably would change after iteration. So at each iteration, filterSQP check the feasibility of the system:

$$\begin{aligned} & \nabla c_j(x_k)^T d + c_j(x_k) \leq 0, \qquad j \in J^{\perp} \\ & ||d||_{\infty} \leq \rho \end{aligned} \tag{10}$$

If the system is satisfied then x^k is a feasible solution, the restoration phase end. Otherwise we have new partition sets $J^{\perp}$ and J and we can continue to solve the SQP subproblem

1.3.4 Sufficient reduction

Clearly, the condition that a new point is not dominated by any entry in the filter allows the possibility of an oscillating sequence of points with accumulation points in (f, h) space that are not Kuhn-Tucker points. A standard way to avoid this in a penalty function algorithm is to require a sufficient reduction in the penalty function on each iteration. A similar idea can be used with a filter algorithm. The aim is to produce an envelope below the filter that prevents points arbitrarily close to the filter from being accepted. This idea is illustrated in Fig. 3 where the envelope is shown by the dashed line.

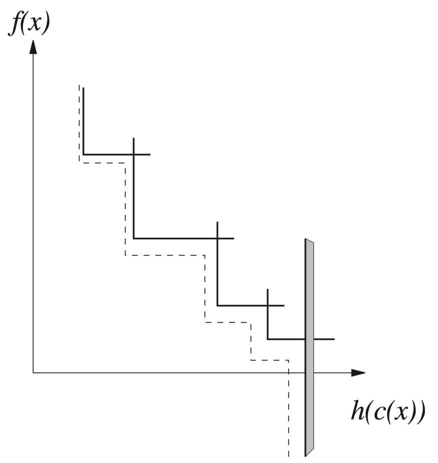


Figure 3: Envelope created by sufficient reduction conditions

A new iterate x^{k+1} is said to be acceptable to the filter, if

$$h^{(k+1)} \leq \beta h^l \tag{11}$$

or

$$f_{k+1} \leq f_l - \max\{\alpha_1 \Delta q_l, \alpha_2 h_l \mu_l\} \tag{12}$$

holds for all filter entries. Here $\beta, \alpha_1, \alpha_2$ are positive constants which can be taken to be $\beta = 0.99, \alpha_1 = 0.25, \alpha_2 = 0.00001$.

1.3.5 Beyond the extreme points of the filter

The current heuristics do not exclude the possibility of generating a sequence of filter entries in which f_k is monotonically increasing and h_k is monotonically decreasing, without converging to a KKT point. To overcome this possibility, a new additional heuristic is introduced as the exact penalty function corresponding to $(f^{(1)}, h(1)) : f^{(1)} + \mu h(1)$. The new point is acceptable when:

$$f_{k+1} + \mu h^{k+1} \leq f^1 + \mu h^1 \tag{13}$$

We refer to this as the North-West corner rule. Hopefully this will be sufficient to avoid the adverse scenario.

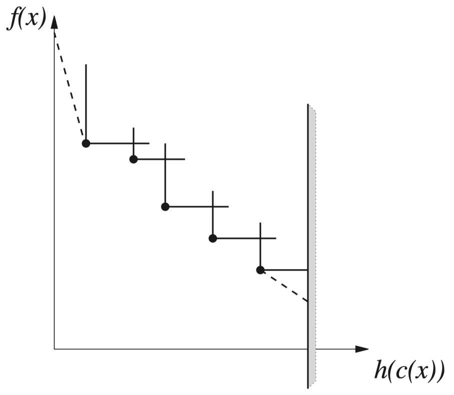


Figure 4: North-West & South-East corner rules

1.4 Numerical results

Fletcher and Leyffer tested the performance of filterSQP and compared to Lancelot. The detailed result can be seen in part 4.3 of Fletcher and Leyffer [2002].

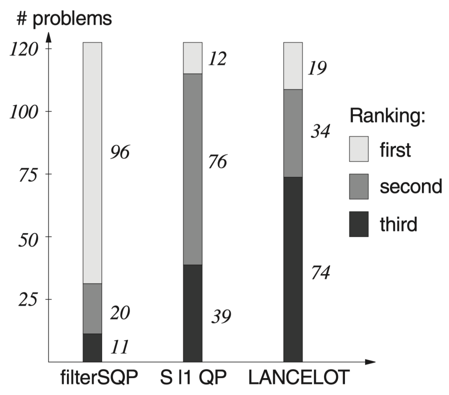


Figure 5: Relative ranking of the three solvers for small test problems

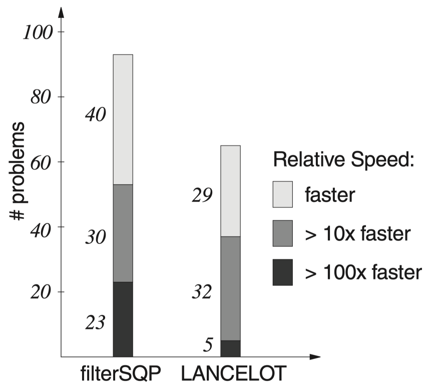


Figure 6: Relative performance of filterSQP and LANCELOT for large test problems

In conclusion, filterSQP is way better than Lancelot for small test problem and also outperform Lancelot in large test problems. The tables are showed in Fig .5, Fig .6.

2 IPOPT

IPOPT, short for "Interior Point OPTimizer, is an open software library for large scale nonlinear optimization. It implements a primal-dual interior point method, and uses line searches based on filter methods. The site of the document is:

`https://coin-or.github.io/Ipopt/`

2.1 Problem formulation

Ipopt solves nonlinear programming of the formWächter and Biegler [2006]:

$$\begin{aligned} \min_{x\in\mathbb{R}^n} \quad & f(x) \\ \text{s.t.} \quad & c(x) = 0 \\ & x^L \leq x \leq x^U \end{aligned} \tag{14}$$

where $x \in \mathbb{R}^n$ are the optimization variable, $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function,

To simplify the notation, in the following discussion we assume the problem is in the following form:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & f(x) \\ & x \geq 0 \end{aligned} \tag{15}$$

2.2 The Primal-Dual Barrier Approach

Barrier methods, based on the earlier work of Fiacco and McCormick, computes the approximate solution for a sequence of barrier problems

$$\begin{aligned} \min_{x} \quad & \varphi_\mu(x) = f(x) - \mu \sum_{i\in\mathcal{I}} \ln(x^i) \\ \text{s.t.} \quad & c(x) = 0 \end{aligned} \tag{16}$$

Then x_*^μ converges to a local solution of (15) as $\mu \to 0$. Consequently, a strategy for solving the original NLP is to solve a sequence of barrier problems for decreasing barrier parameters μ_l, where l is the counter for the sequence of subproblems. The KKT conditions for (16) are

$$\begin{aligned} \nabla\varphi_\mu(x) + A(x)\lambda &= 0 \\ c(x) &= 0 \end{aligned} \tag{17}$$

The system can be solved directly by SQP method, which leads to a so-called primal method. However, the term $\nabla\varphi_\mu(x)$ has components including $\mu/x^{(i)}$, i.e. the system is not defined at a solution $x_*^{(i)} = 0$ of the NLP with an active bound $x_*^{(i)} = 0$, and the radius of convergence of Newton's method applied to the KKT system converges to 0 as $\mu \to 0$

Instead of primal approach, Ipopt use a primal-dual method. Here, we introduce dual variables v, defined as

$$v^i = \frac{\mu}{x^i} \tag{18}$$

With this definition, the KKT conditions are equivalent to the primal-dual equations:

$$\begin{aligned} \nabla\varphi_\mu(x) + A(x)\lambda - v &= 0 \\ c(x) &= 0 \\ x^i v^i - \mu &= 0 \qquad \text{for } i \in \mathcal{I} \end{aligned} \tag{19}$$

Observe that the equations with $\mu = 0$ and the constraints $x, v \geq 0$ are exactly as (15) The primal-dual methods solve the system by Newton-type approach, maintaining iterates for both x_k and v_k, and possibly λ_k. It ensures $x_k^{(i)} > 0$ and $v_k^{(i)} > 0$ for $i \in I$ for all k, and can approach zero only asymptotically as $\mu \to 0$. For this reason, the methods are also often called interior point methods.

We require each barrier problem is solved approximately, so that the solution satisfies

$$||F_{\mu l}(\tilde{x}_{*,l+1}, \tilde{v}_{*,l+1}, \tilde{\lambda}_{*,l+1})||_\infty \leq \varepsilon_l \tag{20}$$

Thus we have the outer loop as Alg 2

Algorithm 2 Outer loop of Ipopt

```
1: initial iteration counter $l = 0$
2: Determine error tolerance $\varepsilon_l$
3: Obtain an approximate solution to the barrier problem for $\mu_l, \varepsilon_l$
4: Update $l = l + 1$ and the barrier parameter $\mu_l$
5: Go back to 2
```

2.3 Computation of Search Direction

We apply Newton's method to (19). Then the search direction $(d_k, d_k^\lambda, d_k^v)$ is obtained by

$$\begin{bmatrix} W_k & A_k & -I \\ A_k^T & 0 & 0 \\ V_k & 0 & X_k \end{bmatrix} \begin{pmatrix} d_k \\ d_k^\lambda \\ d_k^v \end{pmatrix} = - \begin{pmatrix} g_k + A_k\lambda_k - v_k \\ c_k \\ X_k v_k - \mu e \end{pmatrix} \tag{21}$$

Where e is the vector of ones of appropriate dimension, and $Xk := \text{diag}(x_k)$ and $Vk := \text{diag}(v_k)$. W_k denotes the Hessian of the Lagrangian, i.e. $W_k = \nabla_{xx}^2 \mathcal{L}(x_k, \lambda_k, v_k)$ We can obtain a solution of (21) by first solving

$$\begin{bmatrix} H_k & A_k \\ A_k^T & 0 \end{bmatrix} \begin{pmatrix} d_k \\ \lambda_k^+ \end{pmatrix} = - \begin{pmatrix} \nabla\varphi_\mu(x_k) \\ c_k \end{pmatrix} \tag{22}$$

where $H_k = W_k + \Sigma_k$ and $\Sigma_k = X_k^{-1}V_k$, $\lambda^+$is the multipliers of the QP. and then computing d_k^λ and d_k^v from

$$\begin{aligned} d_k^\lambda &= \lambda_k^+ - \lambda_k \\ d_k^v &= \mu X_k^{-1} e - v_k - \Sigma_k d_k \end{aligned} \tag{23}$$

Line search methods usually require that the projection of the Hessian onto some subspace of $\mathbb{R}^n$ is positive definite. Ipopt guarantees that H_k projected onto the null space of A_k^T is positive definite by adding some multiple of the identity, i.e.

$$H_k = W_k + \Sigma_k + \delta_1 I \tag{24}$$

In addition. the iteration matrix in (22) can only be non-singular, if the constraint Jacobian A_k^T has full rank. In order to avoid failure if the condition is not satisfied at some iterate, we add small pivot elements into the lower right corner of the iteration matrix, i.e. (22) is replaced by

$$\begin{bmatrix} H_k & A_k \\ A_k^T & -\delta_2 I \end{bmatrix} \begin{pmatrix} d_k \\ \lambda_k^+ \end{pmatrix} = - \begin{pmatrix} g_k - \mu X_k^{-1} e \\ c_k - \delta_2 \lambda_k \end{pmatrix} \tag{25}$$

and the initial multiplier estimates λ_0 is computed as

$$\begin{bmatrix} I & A_0 \\ A_0^T & 0 \end{bmatrix} \begin{pmatrix} z \\ \lambda_0 \end{pmatrix} = - \begin{pmatrix} \nabla\varphi_\mu(x_0) \\ 0 \end{pmatrix} \tag{26}$$

2.4 Filter based line search method

Ipopt combines the filter method proposed by Fletcher and Leyffer as in the filterSQP. It uses a filter based line search method rather than conventional interior point method using the merit function based line search which combines exact penalty functions and augmented Lagrangian function.

Having computed the direction, a trial point $x_k(\alpha_k, l) := x_k + \alpha_{k,l} d_k^x$ is acceptable if it leads to sufficient progress toward either objective or violation, i.e.

$$\begin{aligned} &\theta(x_k(\alpha_{k,l})) \leq (1 - \gamma_\theta)\theta(x_k) \\ \text{or} \quad &\varphi_\mu(x_k(\alpha_{k,l})) \leq \varphi_\mu(x_k) - \gamma_\varphi \theta(x_k) \end{aligned} \tag{27}$$

$\gamma_\theta, \gamma_\varphi \in (0,1)$ are constants for sufficient reduction. The techniques described in filterSQP are also used in Ipopt filter method, including sufficient reduction, upper bound on constraint violation(Ipopt call it taboo-region), feasibility restoration phase and second order correction.

Assume that the search direction d_k and the maximal step size have been computed from (22). Then back-tracking line search is used to generate a new trail point $x_k(\alpha_k, l) = x_k + \alpha_{k,l} d_k$. Whether the point is acceptable or not is checked by the filter and the Armijo condition has to holds for the proof of convergence.

Thus, the filter algorithm is as Alg. 3

Algorithm 3 Filter based line search in Ipopt

```
   Input: Starting point $x^0$, and the parameters of barrier function and filter method.
1: Initial iteration counter $k = 0$ and the filter $\mathcal{F}_0$
2: Check convergence. If $x_k$ is a local solution of the barrier problem, i.e. if it stratifies the KKT
   conditions, Then the stop.
3: Compute search direction from the linear system (22). If the system is singular, go to Step 9
   feasibility restoration phase.
4: Apply fraction-to-the-boundary rule to compute the maximal step size $\alpha_k^{\max}$
5: Do backtracking line search: Compute new trial point. Check acceptability to the filter. Check
   sufficient decrease. And choose a new step size if necessary.
6: Accept trial point
7: Augment filter
8: Continue with next iteration, i.e. go back to Step 2
9: Compute a new iterate $x_{k+1}$ by restoration phase
```

2.5 Numerical Result

Wächter and Biegler compared Ipopt with other Interior-Point solvers Knitro and LOQO. 954 problems are tested on the same machine. A CPU time limit of 1 hour and an iteration count limit of 3000 was imposed. The tolerance for Knitro and LOQO is 10^{-6} whereas that of Ipopt is set to 10^{-8}.

IPOPT in default mode terminated successfully for 895 out of the 954 problems, whereas only 872 could be solved when the scaling was disabled. Knitro terminated successfully in 829 cases, and LOQO for 847 problems. Fig. 7 presents a performance plot for the iteration count, and Fig. 8 compares the number of function evaluations 9 . Here, 75 problems were excluded because the final objective function values were too different. IPOPT appears to be more efficient in both measures compared to LOQO, and comparable to Knitro in terms of iteration counts. However, Knitro is a trust region method, and the computational costs per iteration are usually not comparable; each unsuccessful trial point in Knitro is counted as one iteration. Looking at Fig. 8, Knitro seems to require overall less function evaluations than IPOPT for the given test set.

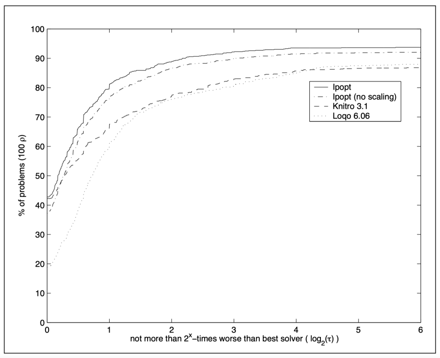


Figure 7: Comparing solvers (iteration count)

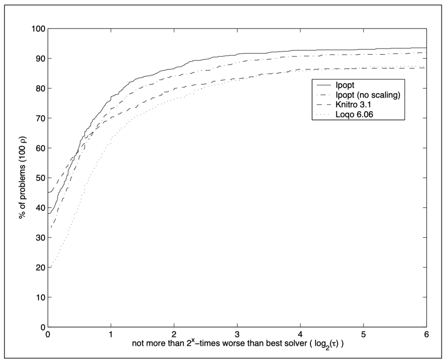


Figure 8: Comparing solvers (function evaluations)

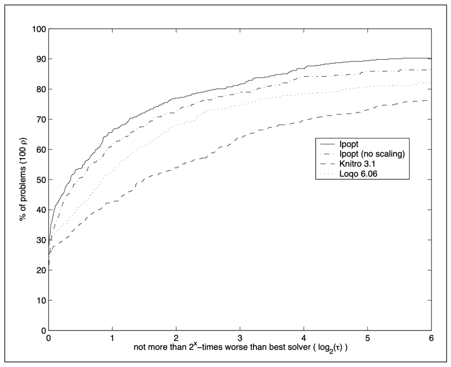


Figure 9: Comparing solvers (CPU time)

Finally, Fig. 9 presents a comparison of the CPU time. Since the CPU time is measured in 0.01s increments on the machine used for obtaining the results, the paper excluded the 444 test problems from the graph, for which the CPU time for the fastest solver was less than 0.05s, as well as 48 additional problems with different final objective function values. As can be seen, Ipopt seems to perform well compared to the other solvers.

3 Knitro

Knitro, short for "Nonlinear Interior point Trust Region Optimization" (the "K" is silent), is a commercial software package for solving large scale nonlinear mathematical optimization problems. It is first released in 2001 as a derivative of academic research at Northwestern University and has since been continually improved by developers at Artelys.

The site of the document is:

`https://www.artelys.com/docs/knitro/`

It characterized by great flexibility and robustness integrating two powerful and complementart algorihmic approaches for nonlinear optimization: the active-set sequential linear-quadratic approach and the interior point appproach. The options are given as KNITRO/ACTIVE. KNITRO/INTERIOR-DIRECT and KNITRO/INTERIOR-CG.

The active-set sequential linear-quadratic programming algorithm is similar in nature to a sequential quadratic programming method, but it uses linear programming sub-problems to estimate the active set at each iteration. This active-set approach may be useful when a good initial point can be provided.

On the other hand, in the interior point methods (see Chapter 17), also known as barrier methods, the nonlinear programming problem is replaced by a sequence of barrier sub-problems controlled by a barrier parameter. The algorithm uses trust-region and a merit function to promote convergence. The algorithm performs one or more minimization steps on each barrier problem and then decreases the barrier parameter and repeats the process until the problem has been solved to the desired accuracy.

3.1 Problem formulation

In KNITRO, the general nonlinear optimization problem is formulated as the following

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & c_E(x) = 0 \\ & c_I(x) = 0 \end{array} \tag{28}$$

where $f : \mathbb{R}^n \to \mathbb{R}, c_E : \mathbb{R}^n \to \mathbb{R}^l$, and $c_I : \mathbb{R}^n \to \mathbb{R}^m$ are twice continuously differentiable functions. Define $E = \{1, \ldots, l\}$ and $I = \{1, \ldots, m\}$.

3.2 KNITRO/ACTIVE

To achieve progress on both feasibility and optimality, the algorithm is designed to reduce the l_1 penalty function:

$$P(x, \sigma) = f(x) + \sigma \sum_{i \in E} |c_i(x)| + \sigma \sum_{i \in I} \max\{0, -c_i(x)\} \tag{29}$$

where the penalty parameter σ is chosen by an adaptive procedure. In the LP phase, given an esimate x_k of the solution of the nonlinear optimization problem (28), the following linear programming problem is solved to estimate the active set:

$$\begin{array}{lll} \min\limits_{p} & \nabla f(x_k)^T p & \\ \text{s.t.} & \nabla c_i(x_k)^T p + c_i(x_k) = 0 & i \in E \\ & \nabla c_i(x_k)^T p + c_i(x_k) \geq 0 & i \in I \\ & ||p||_\infty \leq \Delta_k^{LP} & \end{array} \tag{30}$$

where $\Delta_k^{LP} > 0$ is the trust-region radius.

Since the constraints of (30) may be inconsistent, we solve the l_1 penalty formulation

$$\begin{aligned} l_\sigma(p) = \nabla f(x_k)^T p &+ \sigma_k \sum_{i \in E} |\nabla c_i(x_k)^T p + c_i(x_k)| \\ &+ \sigma_k \sum_{i \in I} \max\{0, -\nabla c_i(x_k)^T p - c_i(x_k)\} \end{aligned} \tag{31}$$

and we solve the following LP problem instead of (30)

$$\begin{aligned} \min_{p} \quad & l_\sigma(p) \\ \text{s.t.} \quad & ||p||_\infty \leq \Delta_k^{LP} \end{aligned} \tag{32}$$

The solution of this linear program, denoted by p^{LP}, is computed by simplex algorithm. Based on this solution, we define the working set W as a independent subset of the active set at the LP solution p^{LP}

We denote the solution of (36) as p^{LP}, which is computed by the simplex algorithm. We can find the active set at the LP solution p^{LP} as $A(p^{LP})$

$$\begin{aligned} A(p^{LP}) = \{i \in E : \nabla c_i(x_k)^T p^{LP} + c_i(x_k) = 0\} \\ \cup\{i \in I : \nabla c_i(x_k)^T p^{LP} + c_i(x_k) = 0\} \end{aligned} \tag{33}$$

To ensure the progress of the algorithm subject to the penalty function defined in (29), define the Cauchy step:

$$p^C = \alpha^{LP} p^{LP} \tag{34}$$

where $\alpha^{LP} \in (0, 1]$ is the step size that provides sufficient decrease in the following piece-wise model of the penalty function $P(x, \sigma)$

$$q_k(p) = l_\sigma(p) + \frac{1}{2} p^T B(x_k, \lambda_k) p \tag{35}$$

Where $B(x_k, \lambda_k)$ is the Hessian of the Lagrange function or an approximation of it. Given the working set W_k for the following equality-constrained quadratic program in variable p is solved, treating the constraints in W_k as equalities and ignoring all other constraints:

Then, given the working set W_k, as a linearly independent subset of the active set $A(p^{LP})$. We can treat the constraints in W_k as equalities and ignoring all other constraints, i.e.

$$\begin{aligned} \min \ & \frac{1}{2} p^T B(x_k, \lambda_k) p + (\nabla f(x_k) + \sigma_k \sum_{i \subset V} \gamma_i \nabla c_i(x_k))^T p \\ \text{s.t.} \ & \nabla c_i(x_k)^T p + c_i(x_k) = 0, \ i \in E \cap W_k, \\ & \nabla c_i(x_k)^T p + c_i(x_k) = 0, \ i \in I \cap W_k, \\ & ||p||_2 \leq \Delta_k \end{aligned} \tag{36}$$

Where γ_i is the algebraic sign of the ith constraint violated in x_k.The solution, denoted by p^Q is obtained by projected conjugate gradient algorithm. The total step p of the SLQP method is computed as

$$p = p^C + \alpha^Q (p^Q - p^C) \tag{37}$$

where $\alpha^Q \in [0, 1]$ is the stepsize.

3.3 KNITRO/INTERIOR

As an interior method, consider the barrier problem to (28)

$$\begin{aligned} \min_{x \in \mathbb{R}^n, s \in \mathbb{R}^m} \quad & f(x) - \mu \sum_{i=1}^{m} \log(s_i) \\ \text{s.t.} \quad & c_E(x) = 0 \\ & c_I(x) = s \end{aligned} \tag{38}$$

where $s \in \mathbb{R}^m$ is a vector of slack variables and $\mu > 0$ is the barrier parameter. The KKT optimality conditions for (38) are :

$$\begin{aligned} & \nabla f(x) - J_E^T(x) y - J_I^T(x) z = 0 \\ & - \mu e + S z = 0 \\ & c_E(x) = 0 \\ & c_I(x) - s = 0 \end{aligned} \tag{39}$$

where $e = [1, \dots, 1]^T, S = \text{diag}(s_1, \dots, s_m)$, $J_E(x)$ and $J_I(x)$ are the Jacobian matrices corresponding to the equality and inequality constraints vectors, respectively.

We also have the following non-differentiable merit function

$$\Phi_\sigma(x, s) = f(x) - \mu \sum_{i=1}^{m} \log(s_i) + \sigma ||c_E(x)||_2 + \sigma ||c_I(x) - s||_2 \tag{40}$$

where $\sigma > 0$.A step is acceptable only if it provides a sufficient decrease of the merit function.

3.3.1 KNITRO/INTERIOR-DIRECT

In this algorithm, the search direction is determined by direct solving of the Newton system associated to the nonlinear system given by the KKT optimality conditions (39)

By applying the Newton method to system (39) in the variables x, s, y, z, we get:

$$\begin{bmatrix} \nabla^2_{xx} L & 0 & -J_E^T(x) & -J_I^T(x) \\ 0 & Z & 0 & S \\ J_E(x) & 0 & 0 & 0 \\ J_I(x) & -I & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ d_y \\ d_z \end{bmatrix} = - \begin{bmatrix} \nabla f(x) - J_E^T(x) y - J_I^T(x) z \\ Sz - \mu e \\ c_E(x) \\ c_I(x) - s \end{bmatrix} \tag{41}$$

If the inertia of the matrix in (41) is

$$(m + n, l + m, 0) \tag{42}$$

then the step d is determined as solution of (41) can be guaranteed to be a descent direction for the merit function (40) We compute the maximum stepsize scalars:

$$\begin{aligned} \alpha_s^{\max} &= \max\{\alpha \in (0, 1] : s + \alpha d_s \geq (1 - \tau) s\} \\ \alpha_z^{\max} &= \max\{\alpha \in (0, 1] : z + \alpha d_z \geq (1 - \tau) z\} \end{aligned} \tag{43}$$

and perform a backtracking line search to compute the stepsize $\alpha_s \in (0, \alpha_s^{\max}], \alpha_z \in (0, \alpha_z^{\max}]$.

The new iterate is computed as

$$\begin{aligned} x &\leftarrow x + \alpha_s d_x, & s &\leftarrow s + \alpha_s d_s \\ y &\leftarrow y + \alpha_z d_y, & z &\leftarrow z + \alpha_z d_z \end{aligned} \tag{44}$$

On the other hand, if the inertia of the matrix in (41) is not as (42), then the step d as a solution of (41) is rejected. In this case, the search direction d is not a descent one for the merit function (40). The algorithm reverts to the trust-region method implemented in KNITRO/INTERIOR-CG.

3.3.2 KNITRO/INTERIOR-CG

The second algorithm implemented in KNITRO computes the search directions using a quadratic model and trust regions. This strategy permits great freedom in the choice of the Hessian and provides a mechanism for coping with Jacobian and Hessian singularities

At the current iterate (x_k, s_k) and for a given value of the barrier parameter μ, first compute the Lagrange multiplier estimates (y_k, z_k), and then the step $d = (d_x, d_s)$ is computed as solution of the following QP sub-problem:

$$\begin{aligned} \min_{d_x, d_s} \quad & \nabla f(x_k)^T d_x + \frac{1}{2} d_x^T \nabla^2_{xx} L(x_k, s_k, y_k, z_k) d_x \\ & - \mu e^T S_k^{-1} d_s + \frac{1}{2} d_s^T \Sigma_k d_s \\ \text{s.t.} \quad & c_E(x_k) + J_E(x_k) d_x = r_E \\ & c_I(x_k) + J_I(x_k) d_x - d_s - s_k = r_I \\ & ||d_x, S_k^{-1} d_s||_2 \leq \Delta_k \\ & d_s \geq -\tau s \end{aligned} \tag{45}$$

where $\Sigma_k = S_k^{-1} Z_k$ and $\tau = 0.995$.

KNITRO uses the null-space approach in which the step d is computed as a sum of a normal step v that attempts to satisfy the linear constraints with $r = 0$ as well as possibly the trust-region and a tangential step that lies on the tangent space of the constraints.

The normal step v is the solution of the following sub-problem:

$$\begin{aligned} \min_{v} \quad & ||J_E v_x + c_E||_2^2 + ||J_I v_x - v_s + c_I - s||_2^2 \\ \text{s.t.} \quad & ||(v_x, S^{-1}v_s)||_2 \leq 0.8\Delta \end{aligned} \tag{46}$$

Once the normal step v has been computed, the vectors r_E and r_I from (45) are computed as the residuals, namely:

$$\begin{aligned} r_E &= J_E v_x + c_E \\ r_I &= J_I v_x - v_s + (c_I - s) \end{aligned} \tag{47}$$

With the normal step $v = (v_x, v_s)$ computed, the sub-problem (45) can be written as

$$\begin{aligned} \min_{d_x, d_s} \; & \nabla f^T d_x - \mu e^T S^{-1} d_s + \frac{1}{2}(d^T \nabla_{xx}^2 L d_x + d_s^T \Sigma d_s) \\ \text{s.t.} \; & J_E d_x = J_E v_x \\ & J_I d_x - d_s = J_I v_x - v_s \\ & ||(d_x, S^{-1}d_s)||_2 \leq \Delta \end{aligned} \tag{48}$$

which is called the tangential sub-problem. Now, to find an approximate solution d of sub-problem (48), firstly introduce the scaling

$$\tilde{d_s} = S^{-1} d_s \tag{49}$$

which transforms (48) into a sphere. Then the projected conjugate gradient (CG) method to the transformed QP is applied, where all iterates are in the linear manifold defined by (48). The iterations are stopped if the boundary of the region is reached or if a negative curvature is detected. Finally, if necessary, the step d is truncated to satisfy (45).

Then the projected conjugate gradient method to the transformed quadratic program is applied, where all iterates are in the linear manifold defined by (52).

4 Benchmark Survey for NLP Solvers

Except the algorithm a solve uses, the performance of each solver depends a lot on the detailed realization. Consequently some newly upgraded solvers combining more techniques and supporting more parallelism would be faster in practice.

A newest collection of comparison can be seen at:

`https://mattmilten.github.io/mittelmann-plots/`

This site contains benchmark for different kinds of problems and compares almost all the solvers. As we can see, the newly published solver Gurobi has outperformed in almost every kind of problems.

However, as a mathematical survey, we care more about the performance of different algorithms (and also combination of algorithms).

In the chapter 21 of Andrei and others [2017], there is comparison with very detailed tables, which compares the performance of MINOS, SNOPT, CONOPT, KNITRO/ACTIVE, KNITRO/INTERIOR, and IPOPT. Part of the results are as following:

In Tab. 1 itt is the total number of iterations, and cput is the total CPU computing time for solving the applications. From Tab. 1 we see that KNITRO and CONOPT are the algorihms with the best performances with respect to the number of iterations and the CPU computing time. The algorithms MINOS and SNOPT have modest performances versus KNITRO, CONOPT, or IPOPT. Observe that the interior point algorithms based on sequential quadratic programming or filter have better performances than MINOS, which is using sequential linear programming or than SNOPT, which is a "pure" sequential quadratic programming algorithm.

Table 1: Global performance of MINOS, SNOPT, CONOPT, KNITRO, and IPOPT

	MINOS	SNOPT	CONOPT	KNITRO	IPOPT
itt	1063	829	798	468	749
cput	347.980	195.810	11.055	18.728	43.094

According to all the results in the book, we can concludes:

1. Both CONOPT and KNITRO are the most robust algorithm. Each of them solve 98 applications fromm 100 considered. On the other hand, IPOPT can solve only 96 applications from 100.

2. In comparison with CONOPT and IPOPT, with respect to the number of iterations and to CPU computing time metrics, KNITRO is the most efficient algorithm. The second in order is IPOPT.

4.1 Analysis of the result

Each of method uses different computational structures to satisfy KKT optimality conditions. Some of these computational structures are able to get the essence of the optimality conditions, thus generating efficient and robust optimization algorithms. These include the augmented Lagrangian, the sequential quadratic programming, and the interior point methods.

Mainly, the augmented Lagrangian and the sequential quadratic programming methods suggest generating and solving a sequence of linear constrained optimization sub-problems with simple bounds on variables. On the other hand, the interior point methods solve a sequence of nonlinear algebraic systems of equations by the Newton method. This is the motivation why the interior point methods are more efficient and more robust versus the augmented (modified) Lagrangian and the sequential quadratic programming methods.

The most powerful nonlinear optimization algorithms combine different optimization techniques and include advanced computational linear algebra techniques. For example, the augmented Lagrangian combined with the penalty functions lead to SPENBAR and to MINOS. The combination of the sequential quadratic programming with the penalty functions generate the algorithms DONLP, NLPQLP, KNITRO, CONOPT. On the other hand, the interior point methods combined with the line search and with the filters generate the very efficient and robust nonlinear optimization algorithms KNITRO-INTERIOR and IPOPT.

Among these combined methods, KNITRO proves to be the most efficient and the most robust algorithm for solving a large variety of nonlinear optimization problems. KNITRO implements the crossover technique which comes from linear programming, another very important ingredient. The idea is to switch to the active-set algorithm as soon as the interior point algorithm has generated an approximate solution by solving the KKT nonlinear system using the direct or the conjugate gradient methods.

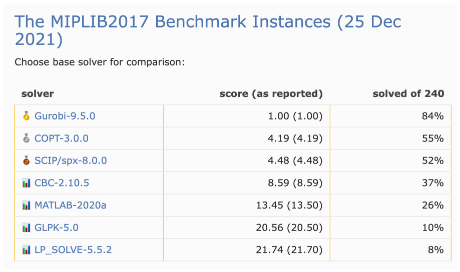
5 Gurobi-MILP

Differed from NLP solvers, almost every solver that works today uses the same algorithm framework. Consequently, the performance of an MILP depends heavily on the realization and combination of techniques.

Table 2: Solvers for MIP/MILP in the order of release time

Solver	Country	Release Time	Solver	Country	Release Time
XPRESS	UK	1983	CPLEX	US	1988
GLPK	Russia	2000	MOSEK	Denmark	2000
SYMPHONY	US	2000	SCIP	Germany	2001
LPSOLVE	Finland	2004	SAS	US	2004
CBC	US	2005	GUROBI	US	2009
MATLAB	US	2014	MIPCL	Russia	2015

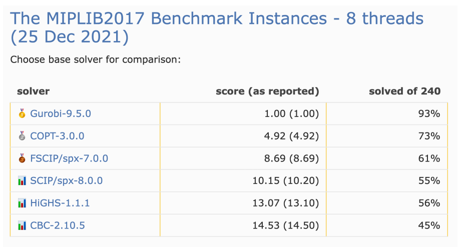
The solvers for MILP is listed in Tab. 2 in the order of the release time. Most of them are still upgrading and thus still works well.


The MIPLIB2017 Benchmark Instances (25 Dec 2021)

Choose base solver for comparison:

solver	score (as reported)	solved of 240
Gurobi-9.5.0	1.00 (1.00)	84%
COPT-3.0.0	4.19 (4.19)	55%
SCIP/spx-8.0.0	4.48 (4.48)	52%
CBC-2.10.5	8.59 (8.59)	37%
MATLAB-2020a	13.45 (13.50)	26%
GLPK-5.0	20.56 (20.50)	10%
LP_SOLVE-5.5.2	21.74 (21.70)	8%

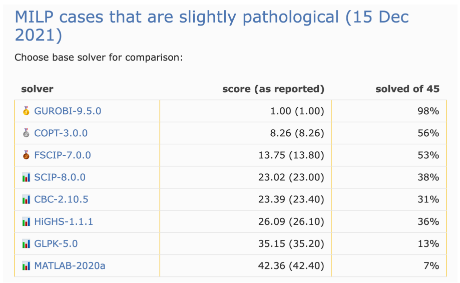
Figure 10: MIPLIB2017 Benchmark


The MIPLIB2017 Benchmark Instances - 8 threads (25 Dec 2021)

Choose base solver for comparison:

solver	score (as reported)	solved of 240
Gurobi-9.5.0	1.00 (1.00)	93%
COPT-3.0.0	4.92 (4.92)	73%
FSCIP/spx-7.0.0	8.69 (8.69)	61%
SCIP/spx-8.0.0	10.15 (10.20)	55%
HiGHS-1.1.1	13.07 (13.10)	56%
CBC-2.10.5	14.53 (14.50)	45%

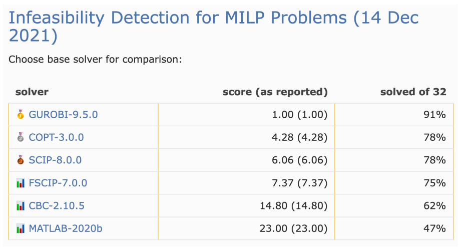
Figure 11: MIPLIB2017 Benchmark with 8 threads


MILP cases that are slightly pathological (15 Dec 2021)

Choose base solver for comparison:

solver	score (as reported)	solved of 45
GUROBI-9.5.0	1.00 (1.00)	98%
COPT-3.0.0	8.26 (8.26)	56%
FSCIP-7.0.0	13.75 (13.80)	53%
SCIP-8.0.0	23.02 (23.00)	38%
CBC-2.10.5	23.39 (23.40)	31%
HiGHS-1.1.1	26.09 (26.10)	36%
GLPK-5.0	35.15 (35.20)	13%
MATLAB-2020a	42.36 (42.40)	7%

Figure 12: MILP cases that are slightly pathological


Infeasibility Detection for MILP Problems (14 Dec 2021)

Choose base solver for comparison:

solver	score (as reported)	solved of 32
GUROBI-9.5.0	1.00 (1.00)	91%
COPT-3.0.0	4.28 (4.28)	78%
SCIP-8.0.0	6.06 (6.06)	78%
FSCIP-7.0.0	7.37 (7.37)	75%
CBC-2.10.5	14.80 (14.80)	62%
MATLAB-2020b	23.00 (23.00)	47%

Figure 13: Infeasibility Detection for MILP Problems

As the latest benchmark shows in Fig. 10, Fig. 11, Fig. 12 and Fig. 13, Gurobi, a relatively newly released solver for business performs best among all the MILP solvers.

More details are available in the open project Visualizations of Mittelmann benchmarks (`https://mattmilten.github.io/mittelmann-plots/`)

5.1 Problem formulation

An MILP problem is formulated as the following form

$$\begin{aligned}
\min \quad & cx \\
\text{s.t.} \quad & Ax = b(\text{linear constraints}) \\
& l \leq x \leq u(\text{bound constraints}) \\
& \text{some or all } x_j \text{ must take integer values}(\text{integrality constraints})
\end{aligned} \tag{50}$$

5.2 Fundamental algorithms

Almost all the MILP solvers share the same Branch-and-Cut algorithm basing on Branch-and-Bound and cutting plane method, here we introduce the two important algorithms.

5.2.1 Branch-and-Bound

As introduced in project 1, Branch-and-Bound is a general method to solve MILP. The basic idea is to solve the problem by linear relaxation and bisection with upper bound to cut branches. A view of problem tree is showed in Fig. 14 The algorithm framework is showed in Alg. 4.

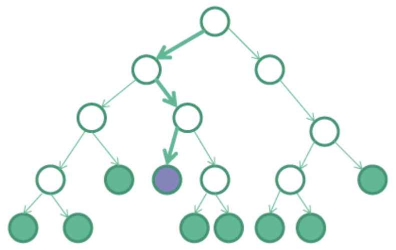


Figure 14: Branch-and-Bound: Each node in the tree is a new subproblem MILP

Algorithm 4 Basic Branch-and-Bound

```
    Input: A MILP Problem $P$, an upper bound $z^-$ of $P$ and corresponding feasible solution $x^-$
 1: Solve the LP relaxation denoted by $P_0$
 2: If $P_0$ has no solution or the optimal solution $x_0$ satisfies integrality constraints, the algorithm is
    done with infeasibility detected or a optimal solution.
 3: Otherwise choose a non-integer variable $x_i$ in the optimal solution of $P_0$ as value $b_i$, split the
    original problem into two subproblem (branches) which add a constraint to split the feasible
    region into $x \leq \lfloor b_i \rfloor$ and $x \geq \lfloor b_i \rfloor + 1$
 4: while Not all branches removed do
 5:     Choose a branch (node of subproblem tree)
 6:     Solve the LP relaxation of the subproblem, denoted by $P_i$
 7:     (a)If $P_i$ has no feasible solution, then we can remove this branch
 8:     (b)If $P_i$ has a solution while it's optimal value is greater than the upper bound $z^-$, since the
    LP relaxation gives the lower bound of $P_i$, so this branch can be removed.
 9:     (c)If the optimal solution $x_i$ meets all the integrality constraints, then $z^-$ is updated as
    min$\{z_i, z^-\}$, also the solution $x^-$ is saved if $z_i < z^-$
10:     (d)If the optimal solution of $P_i$ doesn't meet all the integrality constraints, then we can again
    choose a non-integer variable and split into two branches.
11: end while
```

5.2.2 Cutting lanes method

Conventional Branch-and-Bound works for all the MILP. However, it is not good for large-scale MILP since the problem tree may grow too big and thus the solving time may be too long. Instead Modern MILP solvers takes Branch-and-Cut based on Branch-and-Bound, which add cutting plane method into Branch-and-Bound.

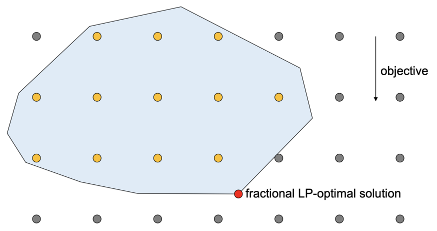


Figure 15: feasible region and LP solution without cutting plane

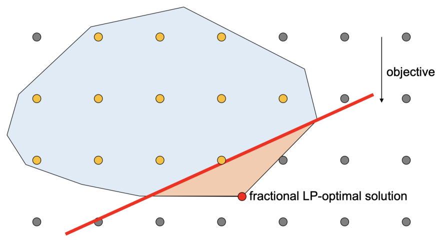


Figure 16: feasible region after cutting plane

As showed in Fig. 15, the feasible region of LP relaxation is bigger than the convex hull of the feasible region of the original MILP problem. Thus if we can cut some non-integer feasible region, it may help solving the MILP problem. An extreme case is that if we cut the plane all the way to the convex hull of the feasible region of the original problem, then we only need to call simplex method to solve the problem.

After Gomory proposed cutting plane method in 1950s, many new cutting methods are developed. The detailed cutting plane methods will be talked later.

5.3 Some important techniques used in Gurobi

The field of mixed integer programming has witnessed remarkable improvements in recent years. Four of the biggest contributors are presolve, cutting planes, heuristics, and parallelism. In addition, good branching variable selection method is a key step which also contribute to Branch-and-Cut.

5.3.1 Presolve

Presolve refers to a collection of problem reductions that are typically applied in advance of the start of the branch-and-bound procedure. These reductions are intended to reduce the size of the problem (both size of variables and constraints), to tighten its formulation, and to identify the problem sub-structures during presolve

Gurobi includes Single-Row Reductions, Single-Column Reductions, Multi-Row Reductions, Multi-Column Reductions and Full Problem Reductions for MIP. As for MILP problem, LP Presolve is explicit and easy to realize. The key idea is to remove redundant variables and constraints by variable substitution.

For example, several constraints are formulated as

$$x + y + z \leq 5 \tag{51a}$$

$$u - x - z = 0 \tag{51b}$$

$$0 \leq x, y, z \leq 1 \tag{51c}$$

$$u \text{ is free} \tag{51d}$$

Then, from (51c) we see that $x + y + z \leq 3$, so (51a) is redundant. From (51b) and (51d) we see that u can be substituted by $x + z$, so (51d) and u are redundant.

5.3.2 Cutting planes

Gomory firstly proposed cutting plane method in 1950s. His cutting method is named as Gomory Cut. The details has been showed in my project 1. The basic idea is to use the structure of simplex method to generate a new constraint from a non-integer simplex method solution.

As a strong MILP solver, Gurobi includes many general cutting plane methods, including Gomory cuts, Mixed Integer Rounding cuts, Flow cover cuts, Lift-and-project cuts, Zero-half and mod-k cuts. Many structural cuts are also included, like Implied bound cuts, Knapsack cover cuts, GUB cover cuts, Clique cuts, Multi-commodity-flow (MCF) cuts, Flow path cuts...

We take some of them as examples (except Gomory cuts which has been introduced):

Flow Cover Cut

Given the system $X = \{(x, y) \in R^n_+ \times B^n : \sum_{j\in N^+} x_j - \sum_{j\in N^-} x_j \leq d, x_j \leq m_j y_j, j \in N\}$ where $N = N^+ \cup N^-$ and $n = |N|$

The x variables are flows that are constrained by the conservation inequality with demand for d and upper bound constraints, where m_j is the capacity of j and y_j is a binary variable that determine if j is open. The flow cover inequality is defined as:

$$0 \leq d - \sum_{x\in C^+} x_j - \sum_{j\in C^{++}} (m_j - l)(1 - y_j) \tag{52}$$

where $l = \sum_{x\in C^+} x_j - d$ and $C^{++} = \{j \in C^+ : m_j > l\}$

Mixed Integer Rounding Cuts

Mixed-Integer Rounding (MIR) cuts can be considered a generalization of the Chvatal integer rounding inequality applied to mixed integer linear programs.

Given $X^{MIR} = \{(x, y) \in Z^p_+ \times R^q_+ : ax + dy \leq b\}$

Then the inequality is given:

$$\sum(\lfloor a_i \rfloor + \frac{\max\{f_i - f_0, 0\}}{1 - f_0})x_i + \sum(\frac{\min\{d_j, 0\}}{1 - f_0})y_j \leq \lfloor b \rfloor \tag{53}$$

where $f_i = a_i - \lfloor a_i \rfloor, f_0 = b - \lfloor b \rfloor$

5.3.3 Branching variable selection

In Branch-and-Bound, to branch we need to choose a variable to branch on. If we select variable arbitrarily, the algorithm might converge slowly. Choice is crucial for the size of the overall search tree.

However, It's expensive to predict which branches lead to infeasibility or big objective moves. (Strong branching :solve for each potential branching variable)

So we need a quick estimate of each branch. Pseudo-Cost is proposed as an idea of using historical data to predict impact of a branch.

Similar to the idea of gradient, the object decrease per change of variable value is computed for each branch. Namely, record pseudo-cost cost$(x_j) = \Delta obj/\Delta x_j$ for each branch. The results are stored in a pseudo-cost table with two entries per integer variable: Average down cost and Average up cost.

When we need to select the variable, we use pseudo cost to predict which variable has more impact and select it.

The numerical results done by Gurobi shows that with Pseudo cost prediction rather than purely random selection, the speed improved 26 times which shows the significant importance of branching variable selection

5.3.4 Heuristics

Many heuristics are proposed in recent years which try to find good integer feasible solution quickly or to improve a given feasible solution, which can contributes a lot to Branch-and-Cut.

Gurobi includes:

Start heuristics

Rounding heuristics: round LP solution to integer values

Fix-and-dive: fix variables, resolve LP

Feasibility pump: push LP solution towards integrality by modifying objective

RENS: Solve sub-MIP in neighborhood of LP solution

Improving heuristics

1-Opt and 2-Opt: Modify one or two variables to get better objective

Local Branching: Solve sub-MIP in neighborhood of MIP solution

Mutation: Solve sub-MIP in neighborhood of 2 or more MIP solutions

RINS: Solve sub-MIP in neighborhood of LP and MIP solution

However, since heuristics also need time, so heuristics would not accelerate too much. Nevertheless, the numerical result provided by Gurobi shows that with option HEURISTICS ON, the time to optimal improves from 90s to 78.9s, and the time to the first feasible solution improves from 35.3 to 9.3 which shows large improvement.

5.3.5 Parallelism

With the improvement of multi-core cpu, Parallelism shows great importance on the speed of a solver. There are several parts in Branch-and-Cut that can run in parallel,. As we can see the main part is that different nodes in MIP tree search can be processed independently. So models that explore large search tree can exploit multi-cores quite effectively and get a constant number of improvement with respect to the number of cpu cores.

5.4 Final Framework

After introducing Branch-and-Bound, cutting planes, presolve, Branching variable selection and heuristics, the algorithm framework of Gurobi for MILP is given in Fig. 17

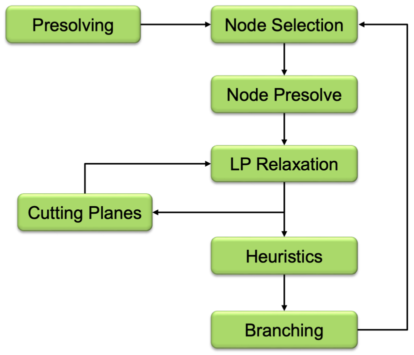


Figure 17: Framework of Gurobi for MILP

References

Neculai Andrei et al. *Continuous nonlinear optimization for engineering applications in GAMS technology*, volume 121. Springer, 2017.

Anthony V Fiacco and Garth P McCormick. *Nonlinear programming: sequential unconstrained minimization techniques*. SIAM, 1990.

Roger Fletcher and Sven Leyffer. Nonlinear programming without a penalty function. *Mathematical programming*, 91(2):239–269, 2002.

Ralph Gomory. An algorithm for the mixed integer problem. Technical report, RAND CORP SANTA MONICA CA, 1960.

Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.